

What Is Claimed Is:

- 1 1. A method for selectively unmarking load-marked cache lines
- 2 during transactional execution, wherein load-marked cache lines are monitored
- 3 during transactional execution to detect interfering accesses from other threads,
- 4 the method comprising:
 - 5 encountering a release instruction during transactional execution of a
 - 6 block of instructions within a program, wherein changes made during the
 - 7 transactional execution are not committed to the architectural state of a processor
 - 8 until the transactional execution completes without encountering an interfering
 - 9 data access from another thread; and
 - 10 in response to the release instruction, modifying the state of cache lines,
 - 11 which are specially load-marked to indicate they can be released from monitoring,
 - 12 to account for the release instruction being encountered;
 - 13 wherein modifying the specially load-marked cache lines can cause the
 - 14 specially load-marked cache lines to become unmarked.
- 1 2. The method of claim 1,
 - 2 wherein a specially load-marked cache line contains a release value
 - 3 indicating how many release instructions need to be encountered before the cache
 - 4 line can become unmarked;
 - 5 wherein modifying the specially load-marked cache line involves
 - 6 decrementing the release value; and
 - 7 wherein if decrementing the release value causes the release value to
 - 8 become zero, the cache line becomes unmarked.

1 3. The method of claim 2, wherein decrementing release values for all
2 specially load-marked cache lines involves incrementing a global counter, wherein
3 release values are initialized with respect to the global counter.

1 4. The method of claim 1, wherein upon encountering a load
2 instruction during the transactional execution, the method further comprises:
3 performing the corresponding load operation; and
4 if the load instruction is a monitored load instruction, load-marking a
5 corresponding cache line to facilitate subsequent detection of an interfering data
6 access to the cache line from another thread;
7 wherein if the load instruction additionally specifies that the corresponding
8 cache line can be released from monitoring during transactional execution, load-
9 marking the cache line involves specially load-marking the cache line to indicate
10 that the cache line can be released from monitoring after either an explicit or
11 implicit number of release instructions have been encountered.

1 5. The method of claim 1, wherein if an interfering data access from
2 another thread is encountered during transactional execution, the method further
3 comprises:
4 discarding changes made during the transactional execution; and
5 attempting to re-execute the block of instructions.

1 6. The method of claim 1, wherein if transactional execution
2 completes without encountering an interfering data access from another thread,
3 the method further comprises:
4 committing changes made during the transactional execution to the
5 architectural state of a processor; and

6 resuming normal non-transactional execution of the program past the
7 block of instructions.

1 7. The method of claim 1, wherein an interfering data access can
2 include:

3 a store by another thread to a cache line that has been load-marked by a
4 thread; and

5 a load or a store by another thread to a cache line that has been store-
6 marked by the thread.

1 8. An apparatus that selectively unmarks load-marked cache lines
2 during transactional execution, wherein load-marked cache lines are monitored
3 during transactional execution to detect interfering accesses from other threads,
4 the apparatus comprising:

5 an execution mechanism configured to execute a release instruction during
6 transactional execution of a block of instructions within a program, wherein
7 changes made during the transactional execution are not committed to the
8 architectural state of a processor until the transactional execution completes
9 without encountering an interfering data access from another thread; and

10 an updating mechanism, wherein in response to the release instruction, the
11 updating mechanism is configured to modify the state of cache lines, which are
12 specially load-marked to indicate they can be released from monitoring, to
13 account for the release instruction being encountered;

14 wherein while modifying the specially load-marked cache lines, the
15 updating mechanism can cause the specially load-marked cache lines to become
16 unmarked.

1 9. The apparatus of claim 8,
2 wherein a specially load-marked cache line contains a release value
3 indicating how many release instructions need to be encountered before the cache
4 line can become unmarked;
5 wherein while modifying the specially load-marked cache line, the
6 updating mechanism is configured to decrement the release value; and
7 wherein if decrementing the release value causes the release value to
8 become zero, the cache line becomes unmarked.

1 10. The apparatus of claim 9, wherein decrementing release values for
2 all specially load-marked cache lines involves incrementing a global counter,
3 wherein release values are initialized with respect to the global counter.

1 11. The apparatus of claim 8, further comprising a load-marking
2 mechanism, wherein upon encountering a load instruction during the transactional
3 execution, the load-marking mechanism is configured to:
4 perform the corresponding load operation; and
5 if the load instruction is a monitored load instruction, to load-mark a
6 corresponding cache line to facilitate subsequent detection of an interfering data
7 access to the cache line from another thread;
8 wherein if the load instruction additionally specifies that the corresponding
9 cache line can be released from monitoring during transactional execution, the
10 load-marking mechanism is configured to specially load-mark the cache line to
11 indicate that the cache line can be released from monitoring after either an explicit
12 or implicit number of release instructions have been encountered.

1 12. The apparatus of claim 8, wherein if an interfering data access
2 from another thread is encountered during transactional execution, the execution
3 mechanism is configured to:

4 discard changes made during the transactional execution; and to
5 attempt to re-execute the block of instructions.

1 13. The apparatus of claim 8, wherein if transactional execution
2 completes without encountering an interfering data access from another thread,
3 the execution mechanism is configured to:

4 commit changes made during the transactional execution to the
5 architectural state of the processor; and to
6 resume normal non-transactional execution of the program past the block
7 of instructions.

1 14. The apparatus of claim 8, wherein an interfering data access can
2 include:

3 a store by another thread to a cache line that has been load-marked by a
4 thread; and
5 a load or a store by another thread to a cache line that has been store-
6 marked by the thread.

1 15. A method for selectively unmarking load-marked cache lines
2 during transactional execution, wherein load-marked cache lines are monitored
3 during transactional execution to detect interfering accesses from other threads,
4 the method comprising:

5 encountering a commit-and-start-new-transaction instruction during
6 transactional execution of a block of instructions within a program, wherein

7 changes made during the transactional execution are not committed to the
8 architectural state of a processor until the transactional execution completes
9 without encountering an interfering data access from another thread; and
10 in response to the commit-and-start-new-transaction instruction,
11 modifying load-marked cache lines to account for the commit-and-start-new-
12 transaction instruction being encountered;
13 wherein modifying the load-marked cache lines causes normally load-
14 marked cache lines to become unmarked, while other specially load-marked cache
15 lines may remain load-marked past the commit-and-start-new-transaction
16 instruction;
17 whereby a new transaction can be immediately started with some cache
18 lines already load-marked, thereby allowing a cache line to be monitored across
19 consecutive transactions.

1 16. The method of claim 15,
2 wherein a specially load-marked cache line contains a checkpoint value
3 indicating how many checkpoint-and-commit instructions need to be encountered
4 before the cache line can become unmarked;
5 wherein modifying the specially load-marked cache line involves
6 decrementing the checkpoint value; and
7 wherein if decrementing the checkpoint value causes the checkpoint value
8 to become zero, the cache line becomes unmarked.

1 17. The method of claim 16, wherein decrementing release values for
2 all specially load-marked cache lines involves incrementing a global counter,
3 wherein release values are initialized with respect to the global counter.

1 18. The method of claim 15, wherein upon encountering a load
2 instruction during the transactional execution, the method further comprises:
3 performing the corresponding load operation; and
4 if the load instruction is a monitored load instruction, load-marking a
5 corresponding cache line to facilitate subsequent detection of an interfering data
6 access to the cache line from another thread;
7 wherein if the load instruction additionally specifies that multiple
8 checkpoint-and-commit instructions need to be encountered before the cache line
9 can become unmarked, and load-marking the cache line involves specially load-
10 marking the cache line to indicate that multiple checkpoint-and-commit
11 instructions need to be encountered before the cache line can become unmarked.

1 19. The method of claim 15, wherein if an interfering data access from
2 another thread is encountered during transactional execution, the method further
3 comprises:
4 determining if the transactional execution was initiated by a commit-and-
5 start-new-transaction, and if so executing alternative code to complete the
6 transactional execution in the presence of the interfering data access; and
7 otherwise, discarding changes made during the transactional execution, and
8 attempting to re-execute the block of instructions.

1 20. The method of claim 15, wherein if transactional execution
2 completes without encountering an interfering data access from another thread,
3 the method further comprises:
4 committing changes made during the transactional execution to the
5 architectural state of the processor; and

6 resuming normal non-transactional execution of the program past the
7 block of instructions.

1 21. The method of claim 15, wherein an interfering data access can
2 include:

3 a store by another thread to a cache line that has been load-marked by a
4 thread; and

5 a load or a store by another thread to a cache line that has been store-
6 marked by the thread.

1 22. An apparatus that selectively unmarks load-marked cache lines
2 during transactional execution, wherein load-marked cache lines are monitored
3 during transactional execution to detect interfering accesses from other threads,
4 the apparatus comprising:

5 an execution mechanism configured to execute a commit-and-start-new-
6 transaction instruction during transactional execution of a block of instructions
7 within a program, wherein changes made during the transactional execution are
8 not committed to the architectural state of a processor until the transactional
9 execution completes without encountering an interfering data access from another
10 thread; and

11 an updating mechanism, wherein in response to the commit-and-start-new-
12 transaction instruction, the updating mechanism is configured to modify load-
13 marked cache lines to account for the commit-and-start-new-transaction
14 instruction being encountered;

15 wherein while modifying the load-marked cache lines, the updating
16 mechanism can cause normally load-marked cache lines to become unmarked,

17 while other specially load-marked cache lines may remain load-marked past the
18 commit-and-start-new-transaction instruction;
19 whereby a new transaction can be immediately started with some cache
20 lines already load-marked, thereby allowing a cache line to be monitored across
21 consecutive transactions.

1 23. The apparatus of claim 22,
2 wherein a specially load-marked cache line contains a checkpoint value
3 indicating how many checkpoint-and-commit instructions need to be encountered
4 before the cache line can become unmarked;
5 wherein while modifying the specially load-marked cache line, the
6 updating mechanism is configured to decrement the checkpoint value; and
7 wherein if decrementing the checkpoint value causes the checkpoint value
8 to become zero, the cache line becomes unmarked.

1 24. The apparatus of claim 23, wherein decrementing release values
2 for all specially load-marked cache lines involves incrementing a global counter,
3 wherein release values are initialized with respect to the global counter.

1 25. The apparatus of claim 22, further comprising a load-marking
2 mechanism, wherein upon encountering a load instruction during the transactional
3 execution, the load-marking mechanism is configured to:
4 perform the corresponding load operation; and
5 if the load instruction is a monitored load instruction, to load-mark a
6 corresponding cache line to facilitate subsequent detection of an interfering data
7 access to the cache line from another thread;

8 wherein if the load instruction additionally specifies that multiple
9 checkpoint-and-commit instructions need to be encountered before the cache line
10 can become unmarked, the load-marking mechanism is configured to load-mark
11 the cache line to indicate that multiple checkpoint-and-commit instructions need
12 to be encountered before the cache line can become unmarked.

1 26. The apparatus of claim 22, wherein if an interfering data access
2 from another thread is encountered during transactional execution, the execution
3 mechanism is configured to:

4 determine if the transactional execution was initiated by a commit-and-
5 start-new-transaction, and if so to execute alternative code to complete the
6 transactional execution in the presence of the interfering data access; and
7 otherwise, to discard changes made during the transactional execution, and
8 to attempt to re-execute the block of instructions.

1 27. The apparatus of claim 22, wherein if transactional execution
2 completes without encountering an interfering data access from another thread,
3 the execution mechanism is configured to:

4 commit changes made during the transactional execution to the
5 architectural state of the processor; and to
6 resume normal non-transactional execution of the program past the block
7 of instructions.

1 28. The apparatus of claim 22, wherein an interfering data access can
2 include:
3 a store by another thread to a cache line that has been load-marked by a
4 thread; and

5 a load or a store by another thread to a cache line that has been store-
6 marked by the thread.